DoubleCheck for Oblivious HTTP and beyond

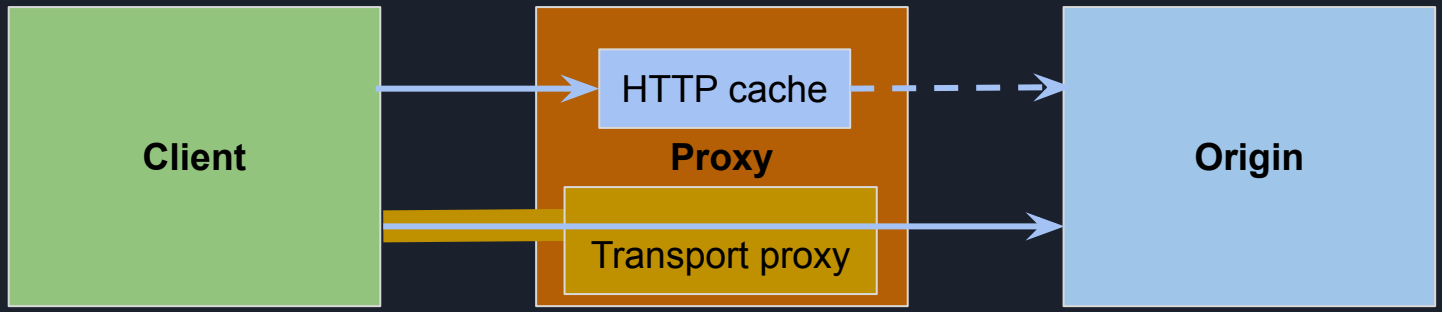
Ben Schwartz, OHAI @ IETF 115

Changes since IETF 114

- Removed:
  - Normative dependencies on OHTTP and Access Service Descriptions
  - Requirement to use CONNECT-UDP and HTTP/3
  - “MUST” use a transport proxy (changed to “SHOULD”)
- Added:
  - Text on how DoubleCheck can be used for Privacy Pass
  - New terminology for the participants’ roles
  - Example showing DoubleCheck for ODoH using draft-pauly-ohai-svcb-config

Reminder: **How DoubleCheck works**

- The **Client** wants a **Desired Resource** on the **Origin**
- The **Client** fetches it through the **Proxy**, acting as a caching HTTP request proxy
  - Guarantees consistency but not authenticity
- The **Client** fetches it again directly from the **Origin**
  - Guarantees authenticity but not consistency
  - SHOULD run the transport through the same **Proxy** to maintain the IP anonymity set
- **Check that the contents match!**



Key Ideas

- If you need Key Consistency, you usually also need a proxy that is trusted not to reveal your identity to the destination.
  - Otherwise, your IP address gives you away.
- You only need Key Consistency with this proxy's other users.
  - Users of other proxies are distinguishable from you anyway.
- HTTP caching rules are all you need to achieve this.
  - …provided the resource's origin sets the right headers.
  - …provided that everyone interprets the HTTP caching rules in a very specific way.

What guidance to provide about leakage

Privacy leakage sources noted in the draft:

- Temporal correlation between check and use
  - “Establish a transport tunnel through the Proxy to the Origin (OPTIONAL)”
  - “If the Proxy offers an Encrypted DNS service, it MUST NOT enable EDNS Client Subnet”
  - “Clients MUST perform each fetch to the Origin … as a fully isolated request. Any state related to this Origin (e.g., ...) MUST NOT be shared with prior or subsequent requests.”
  - “This specification does not offer specific mitigations for protocol fingerprinting.”
- Cache eviction attacks
  - “Proxies SHOULD employ defenses against malicious attempts to fill the cache. Some possible defenses include: …”

Is this how we want to do it?

- *Consistency* alternative: **Bespoke consistency logic**
  - More explicit: Avoids subtle reinterpretations of existing standards (e.g., If-Match, “immutable”)
  - Smoother: Might be able to avoid a “Thundering Herd” of revalidations when a popular resource changes.
  - Requires a lot of new protocols and logic for all three parties
- *Authenticity* alternative: **Object Security (e.g. JSON Web Signature)**
  - Faster: Removes the need for the second check
  - More complicated, less convenient: Origin needs to change its certificate handling, etc.

Appendix

Example: Platform telemetry

- My OS installation image came configured to report telemetry to a default telemetry service that supports OHTTP.
- I believe my OS image is the same as everyone else's, and I trust the code running locally, but **I want to prevent the telemetry service from linking my reports together**.
  - **Otherwise the OHTTP Relay is unnecessary!**
- I have configured my OS with an OHTTP Relay that I trust not to collude with the telemetry service, but **I don't trust the Relay with the contents of my telemetry reports**.
  - **Otherwise the OHTTP Gateway is unnecessary!**
- **Problem**: How do I ensure that the Gateway URL, KeyConfig, and Target URL are authentic and the same as everyone else's?

Easy answer: Hardcode everything!

- Bake the Gateway URL, Target URL, and KeyConfig right into the OS image
  - or in general distribute them through a trusted, consistent “bootstrap channel”.
- **Problem**: This prevents key rotation and other operational adjustments.

Many details

- Stitched together from standard HTTP cache and proxy components
  - Headers used: Cache-Control, ETag, If-Match, Age
  - Some additional requirements beyond general HTTP compliance.
- Defenses against different attackers
  - Malicious Relays (KeyConfig forgery)
  - Malicious Service Description Hosts (cache wiping, fetch timing correlation)
  - Colluding malicious Clients (cache wiping, cache eviction)
- Performance considerations
  - Various recommended optimizations
  - Overall latency is generally 2 RTT through the proxy